Downcasing types
(Introductory notes)

Eduardo Ochs
eduardoochs@gmail.com
http://angg.twu.net/
http://angg.twu.net/math-b.html
http://angg.twu.net/LATEX/2007dnc-sets.{tex,dvi,pdf}

Written: 2007.
LaTeXing and minor additions and revisions: may 2008.
Current version: 2008may20 (see the footer)

Index of the slides:

Main idea: lifting archetypal proofs
Take a proof of some general statement.
Now specialize it to a particular case.
"Specializing" is somehow like doing a
projection - some distinctions collapse,
some details are lost -

Here we are concerned with the opposite
of specializing proofs.
We start with an "archetypal proof" -
a proof of a certain very particular case,
done in a certain language - and then we
change our dictionary a bit - and some
terms change meaning; and the same proof
becomes a proof of the general case -

We call this "lifting". Archetypal proofs
/lift/ to more general proofs, and when this
happens we get general proofs done in a
language inherited from the archetypal cases.

My favorite examples of such liftings of
proofs are for proofs in Categorical
Semantics.

The fist significant example will be this
one (we call it "CCC $\leftrightarrow \lambda 1$"):

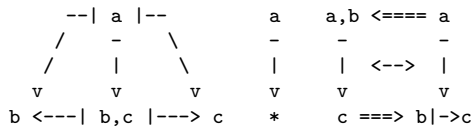
The Cartesian-Closed Categories (CCCs)
are exactly the categories where we can
interpret the simply-typed λ-calculus
("System $\lambda 1$").

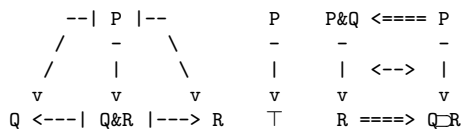
After "CCC $\leftrightarrow \lambda 1$" we will see "Hyp $\leftrightarrow$ LPCE":

The hyperdoctrines are exactly the
categories in which we can interpret
(a certain system of intuitionistic, typed)
first-order logic ("LPCE").

Lifting set-like notation
Our approach:

'Set' is our archetypal CCC.
Sets are also our archetypal model for $\lambda 1$.
Most of the proof - even most of the details
of the statement of the theorem! - will appear
more or less naturally from taking the
diagrams below, done in the language of
"downcased types" (DNC) and translating
them in several ways.





Later, **Set** (or, more precisely, the fibration $\mathbf{Set}^{\rightarrowtail}$),
we will also be our archetypal hyperdoctrine.

Starting point: the Curry-Howard Isomorphism
There is a correspondence between
Natural Deduction trees and
terms of the simply-typed λ-calculus ($\lambda 1$).

Example:

```
        P&Q                      d:A×B
        ---                      -----
P&Q     Q       Q⊃R    d:A×B     π'd:B     f:B->C
---     ----------     -----     ---------------
 P          R          πd:A       f(π'd):C
 -----------           ------------------
    P&R                  <πd,f(π'd)>:A×C
```

Sometimes the usual notation for type theory/
λ-calculus feels too verbose.

The DNC notation started as an informal reasoning tool -
I used it in my private notes as a system of abbreviations
for λ-calculus.

DNC grew with time and started to look more formal...

It got special arrow symbols (a.k.a. "connectives" - '$\Rightarrow$' and
'$\stackrel{\bullet}{\rightarrow}$') to represent functors and natural transformations.

In 2001 I found that '$\Rightarrow$' and '$\stackrel{\bullet}{\rightarrow}$' had introduction rules
besides the obvious elimination rules (think in introd and
elim of '&', '$\supset$', '$\vee$', etc in Natural Deduction) -
so DNC could be the basis for a system of natural deduction
for categories.

DNC never became formal enough to be a "system", and nowadays it
lives a more unpretentious existence as an auxiliary language for
type theory and category theory (esp. categorical semantics).

I changed the meaning for "DNC" -
from a sytem for "Natural Deduction for Categories" to
a notation of "DowNCased types".

Main ideas

```
Long names for variables
------------------------
In {(n,r)∈N×R | r^2=n}
the (n,r) is the name of a variable -
N×R is the space of pairs made of a
natural number and a real.

A dictionary
------------
Define n := π (n,r)
       r := π'(n,r)
   (r,n) := <r,n>

The syntactical distinction between variables and terms
-------------------------------------------------------
We've sacrificed it.
Also, without the dictionary there is no way to tell from
(n,r), n, r, (r,n) which of them are more primitive than
the others.

Downcasing
----------
The default name for a variable ranging over A is 'a'.
The default (long!) name for a variable ranging over A×B is '(a,b)'.

Well-formed formulas, expressions, etc
--------------------------------------
We don't have these notions in DNC.
Anything can be a DNC ''name''. 8-(

Spaces of functions
-------------------
In f:A->B we will read ''A->B'' as B^A
and the '':'' as the ':' of Type Theory.
(For mathematicians: ':' is roughly like '∈', but ''f:A->B''
means also '''f' is of type 'A->B''', and each term has
a single type. We are not interesting in '\subset', '\subseteq', etc).

Notation for functions
----------------------
An 'a|->b' is an element of A->B.
An 'a|->b' is a function that takes each 'a' to a 'b'.
```

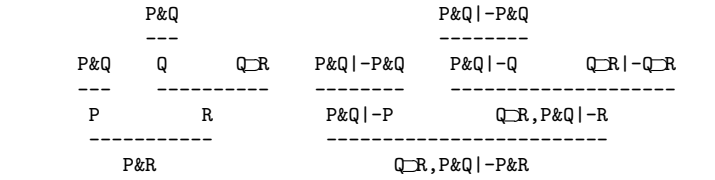
Reconstructing the types and the dictionary from trees

```
          d:A×B
          -----
  d:A×B   π'd:B    f:B->C
  -----   ---------------
  πd:A      f(π'd):C
  ------------------
    <πd,f(π'd)>:A×C

         a,b
         ---
   a,b    b    b|->c      a := π (a,b)
   ---    ----------      b := π'(a,b)
    a         c           c := (b|->c)(b)
    -----------         a,c := <a,c>
       a,c
```

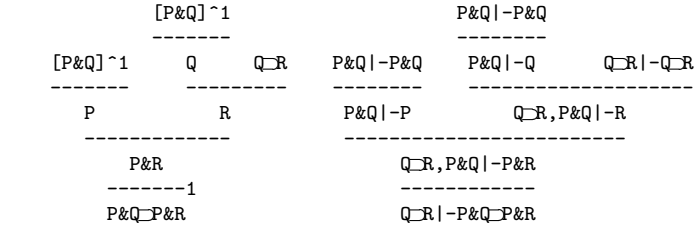
Now add ‘‘a,b := d’’ and ‘‘b|->c := f’’ and we get
the λ-calculus terms back from the DNC tree.
Note that we obtain the types back by uppercasing
the DNC ‘‘terms’’ -
  ‘|->’ is uppercased to ‘->’,
  ‘,’ to ‘×’, etc.

Contexts and discharges

Each subtree of a Natural Deduction tree/derivation/proof corresponds to a smaller proof contained in a bigger one. For each node of a tree let's look at the tree above the node, and make its hypotheses explicit.



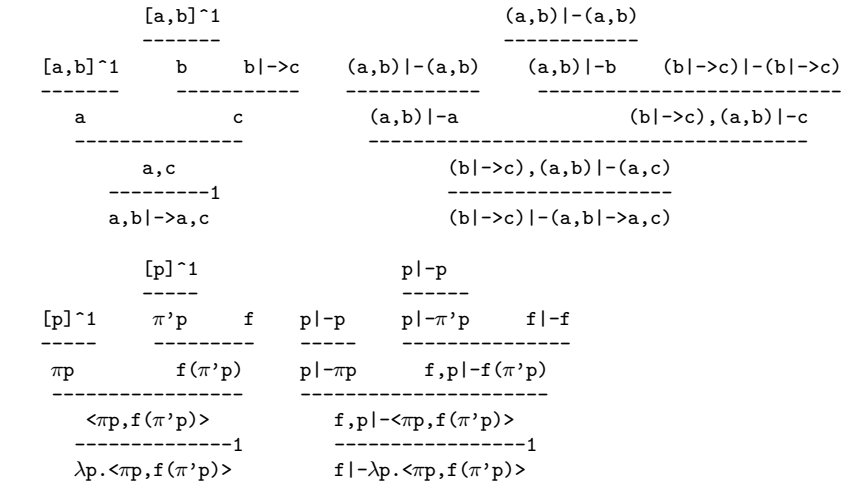
Now we can understand the ''⊃-introduction'' rule; it involves a discharge.



Below the bar marked with ''1'' the hypotheses marked with ''[·]^1'' should no longer be in the list of hypotheses - they've been ''discharged'' (into the conclusion).

Contexts and discharges (2)

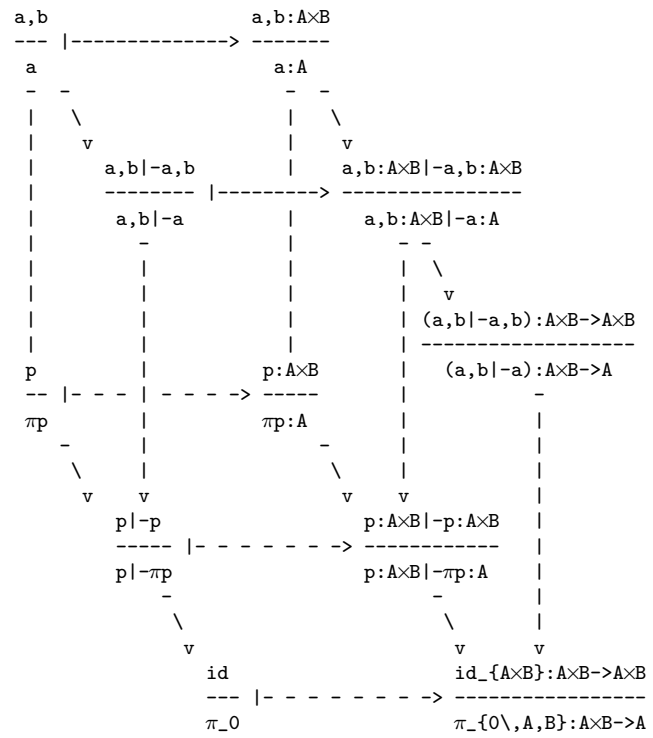
Let's look at the functional side of this (Curry-Howard):



The ''⊃-introduction'' rule (last bar) corresponds to introducing a 'λ';
one variable ceases to be free, and is removed from the list of hypotheses.

One way of reading ''f,p|-<πp,f(π'p)>'' -
or ''f:B->C,p:A×B|-<πp,f(π'p)>:A×C'' - is:
if we know the value of f and p (points of B->C and A×B)
we know the value of <πp,f(π'p)> (a point of A×C).

Expanding trees



Theorems for Free

From the ‘‘Theorems for Free’’ paper (Wadler 1989):
* All closed terms of type ∀A.∀B.(A×B->B×A)
  obey a certain theorem
* Corollary: all terms of type ∀A.∀B.(A×B->B×A)
  are the flip function.

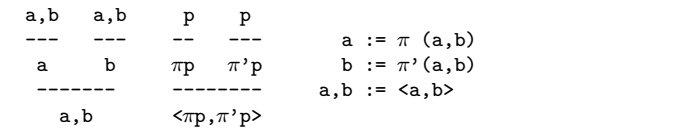
Something similar (but much looser) happens in DNC:
‘a,b|->b,a’ has a ‘‘natural definition’’ -
apply Curry-Howard, prove P&Q⊐Q&P, translate the
proof to λ-calculus in DNC notation -

```
  [P&Q]^1   [P&Q]^1   [a,b]^1   [a,b]^1   [p]^1   [p]^1
  -------   -------   -------   -------   -----   -----
     Q         P         b         a       π'p     πp
     -----------         -----------       ----------
         Q&P                 b,a            <π'p,πp>
       -------1            ---------1       -----------1
       P&Q⊐Q&P             a,b|->b,a        λp.<π'p,πp>
```

and an argument using normalization of ND proofs
can be used to show that all other ND proofs (= λ-terms)
are equivalent to that one.

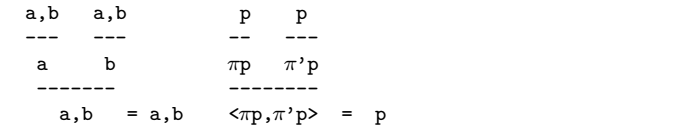
In DNC we won’t be interested in the ‘‘uniqueness’’ part
very often, though.

Ambiguity plays on our side (usually)



Apparently the DNC tree on the left, above introduces a ‘‘secondary definition’’ for a,b and a circularity in the dictionary, and such things have to be avoided at all costs –

But we can be careful and interpret each tree as a term, and in



what happens is that we have two different ‘‘natural constructions’’ for a,b from a,b, and we’re saying that the two ‘‘must give the same result’’, i.e., <πp,π’p> = p...

In many contexts we will have two different natural constructions for a DNC ‘‘term’’ with a certain name, and we will want to say that the two constructions give the same result (sometimes this will be an axiom, sometimes a theorem, sometimes a hypothesis) –

We will sometimes use the expression ‘‘is well defined’’ (notation: ‘‘wd[·]’’) in a funny sense: ‘‘wd[a,b]’’ will mean ‘‘the two obvious natural constructions for a,b give the same result’’.

Informal coherence

A /coherence theorem/ says that
there is exactly one entity
of a given type...

For example: that ‘‘there is
exactly one map A×B->B×A’’ -
the flip function.

Coherence would allow us to say
‘‘_the_ map a,b|->a,b’’ instead of
  ‘‘_a_ map a,b|->a,b’’.

We have something weaker in DNC:
a dictionary, and a vague notion
of assigning ‘‘natural meanings’’
to names.

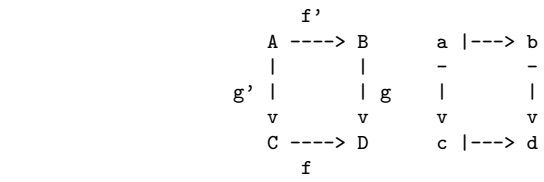
Each ‘‘natural meaning’’ comes
from a ‘‘natural construction’’.

Instead of proving ‘‘real’’ coherence -
e.g., ‘‘all natural constructions for
a,b|->b,a give the same result’’ -
we will use something MUCH weaker:

  ‘‘all the ‘obvious’ natural
    constructions for a,b|->b,a
    (in a given context - given
    some base objects,
    operations, and diagrams)
    yield the same result.’’

Informal coherence: examples

If we have set meanings for the
four arrows in the diagram below,



then we have two different ‘‘natural
constructions’’ for a|->d, each one
giving a (possibly different)
‘‘natural meaning’’ for a|->d...

‘‘wd[\alpha]’’ (pronounced:
‘‘\alpha is well-defined’’)

means:

‘‘all the obvious natural
constructions for \alpha
yield the same result’’

In the diagram above
‘‘\wd[a|->d]’’ means f';g=g';f
(i.e., a|->b|->d = a|->c|->d)
i.e.: ‘‘the square commutes’’.

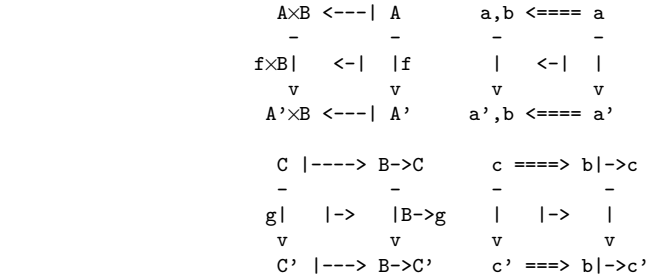
Giving meaning to a ‘‘wd’’ expression
involves deciding which are the
‘‘obvious natural constructions’’.

Informal coherence: examples (2)

```
Each sentence ‘‘wd[\alpha]’’ is a
proposition whose precise meaning
is given as an entry in the dictionary.

Another example: in

  a |---> b |---> c
  -       -       -
  |       |       |
  v       v       v
  d |---> e |---> f
  -       -       -
  |       |       |
  v       v       v
  g |---> h |---> i

we have:

  wd[a|->e] & wd[b|->f] &
  wd[d|->h] & wd[e|->i] => wd[a|->i].

Note that:

  wd[a|->i] := (a|->b|->c|->f|->i =
                a|->b|->e|->f|->i =
                a|->b|->e|->h|->i =
                a|->d|->e|->f|->i =
                a|->d|->e|->h|->i =
                a|->d|->g|->h|->i).

Each of the other ‘wd[·]’s is an equality
between only two composites (not six).
```

Downcasing functors

A functor F:\catC->\catD is composed of two operations:
  F_0 - its action on objects, and
  F_1 - its action of morphisms.
Fix a set B, and look at these diagrams, for the functors
(×B):Set->Set and (B->):Set->Set:



Functors act on two levels, so we will use a double arrow -
'=>' - to denote them. Their names in DNC will come from
their action on objects:

```
A|->A×B     a=>a,b
C|->B->C    c=>b|->c
```

Their ''syntactical actions'' on morphisms can be derived
from their names: a=>a,b adds a '',b'' to the name of each
object, so it takes a|->a' to a,b|->a',b; same for c=>b|->c.

To find the ''real action'' of (×B)=(a=>a,b) on morphisms,
look for a natural construction of a,b|->a',b from a|->a;
f×B = a,b|->a',b = λp.<f(πp),π'p>.

Note that a functor a=>a,b does not take 'a's into 'a,b's,
like an arrow a|->a,b would do; instead, it takes the
whole space of 'a's, E[a]=A, into the space of 'a,b's,
E[a,b]=A×B - (×B)_0 = λA:Sets.A×B.

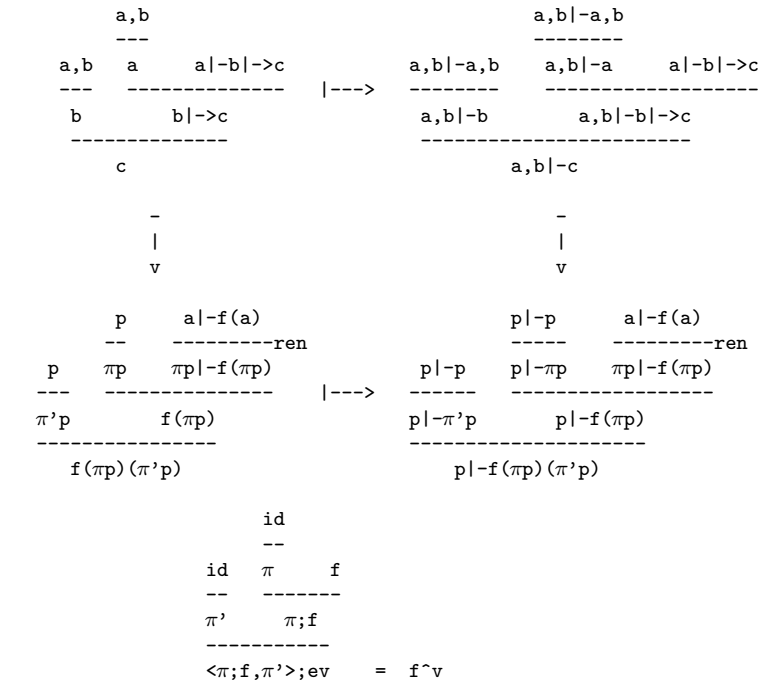
Products via natural transformations

```
Any diagram b <--| p |--> c induces an operation that
takes any object A to a map of sets

   (A->P) -> (A->B)×(A->C)

   / a \      /       a       \
   | - |      |     -   -     |
   | | | |--> |    /     \    |
   | v |      |   v       v   |
   \ p /      \ b           c /
```

Downcasing Natural Transformations

```
Between two contravariant functors,

  Hom(-,P): Set^op -> Set
               A |-> A->P
             a^op => a|->p

  Hom(-,B)×Hom(-,C): Set^op -> Set
                        A |-> (A->B)×(A->C)
                      a^op => (a|->b),(a|->c)

The natural transformation

  T: Hom(-,P) -> Hom(-,B)×Hom(-,C)

takes each object A of Set (or: an object of Set^op)
to a function:

  TA: (A->P) -> ((A->B)×(A->C))

We will represent this natural transformation T in DNC
by just downcasing the triangle:

          A
        - - -
       /  |  \
      v   v   v
   (A->P) -> ((A->B)×(A->C))
```

Translating bigger trees



Downcasing categorical diagrams (in general)

One day I realized that the same notation that I was using for sets did also work - with no changes in the syntax, only in the semantics - for categories.

Here's a comparison; in the second situation, at the right, \catC is an arbitrary category.

A is a set	A is an object of \catC, A∈|\catC|
B is a set	B is an object of \catC, B∈|\catC|
A ---> B (a function)	A ---> B (a morphism)
a |--> b (a function)	a |--> b (a morphism)
A is the space of 'a's	Now:
B is the space of 'b's	''an 'a''' is an abuse of language
A = E[a]	''a 'b''' is an abuse of language
B = E[b]	'a|->b' has semantics - it's a morphism -
	but 'a' and 'b' have no semantics
An 'a' is a point of A,	
a 'b' is a point of B,	'a's and 'b's are not points in this case,
a∈A, b∈B.	just ''pseudopoints'', and pseudopoints
	don't need to exist - they may be just
	syntactical devices
	To enforce the distinction we say
	''object of'' instead of ''space of''
	and we write O[·] instead of E[·]
	A is the objects of 'a's
	B is the objects of 'b's
	A = O[a]
	B = O[b]

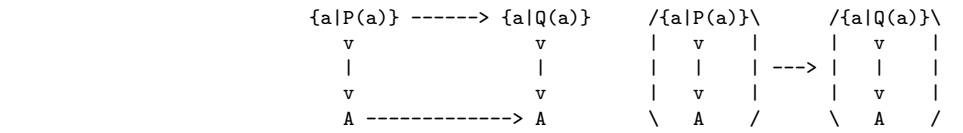
Note: it's almost impossible to understand this is the general case without examples...

Why we want to allow pseudopoints

For example, for subobjects.
Implications can be seen as inclusions.

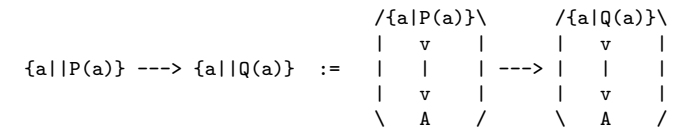
  ∀a∈A.P(a)⊐Q(a)
  {a∈A|P(a)} \subset {a∈A|Q(a)}

Categorically,



This is a morphism between two objects of Set^>->, where
Set^>-> is the category of monic arrows between objects of Set.
Set^>-> is also called Sub(Set) - the category of subojects of Set.

Notational trick:



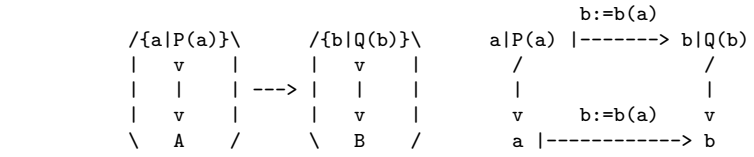
The ‘|’ in {a|P(a)} is called the ‘‘such that’’ bar;
we’re doubling it to represent subobjects.
We downcase that into:

  a||P(a) |-> a||Q(a)

this morphism - in Set^>-> is not just a mapping of points...
in fact, it’s two morphisms in Set - a|P(a) |-> a|Q(a) and a|->a -
plus the information that the square below commutes:

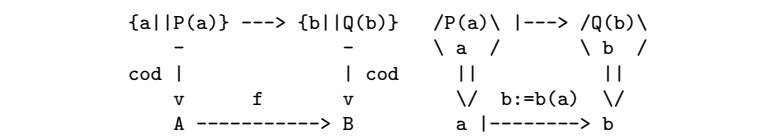
Maps between subobjects: the general case

More generally,
{a||P(a)} -> {b||Q(b)} is:

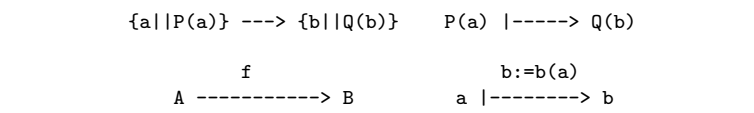


a||P(a) |-> b||Q(b) means ∀a.P(a)⊃Q(b(a)).
Note the use of a bit of ‘‘physicist’s notation’’ here -
in the presence of a function f = a|->b the value of b can
be seen as a function of a, and a physicist would write
b=f(a) or b=b(a)... we will prefer b=b(a) to avoid using
new letters, and the dictionary will translate such
expressions (atrocities!) into something mathematically
sound.

We will usually write pseudopoints like a||P(a)
as ‘‘P(a) over a’’ (in parentheses), and we will draw these
things over their ‘‘codomain projections’’:

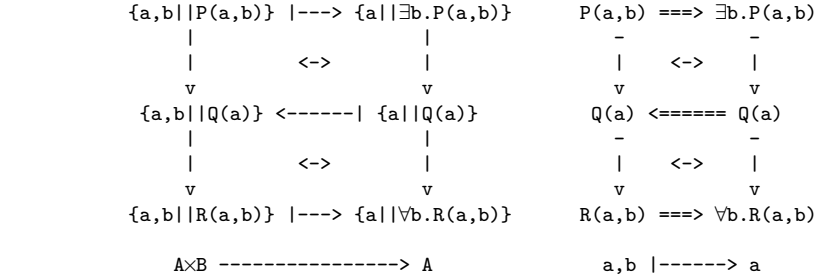


In some rare occasions, when space is at premium, we will
write just P(a) instead of ‘‘P(a) over a’’.
The ‘‘codomain projection’’ functor arrows will usually
not be draw.



Quantifiers, categorically

In Set^>-> the functors that quantify over a variable
are adjoints to ‘‘weakening functors’’ that add a dummy
variable to the domain:



These functors do not operate on the whole of Set^>-> -
they go from one ‘‘fiber’’ of Set^>-> to another.

The best way to formalize this is via /fibrations/;
Set^>-> is a fibration over Set.

A fibration is composed of:

an ‘‘entire category’’	Set^>->
a ‘‘base category’’	Set
a ‘‘projection functor’’	Cod: Set^>-> -> Set
and enough ‘‘cartesian liftings’’.	(?!?!?! - next slide)

In our diagrams we will not usually draw the projection functors.
Instead we will just draw the objects of the entire category
over their projections - and we will often omit the part
of their names that can be recovered from the projections.
In the diagram above we wrote just ‘Q(a)’, not ‘a||Q(a)’.

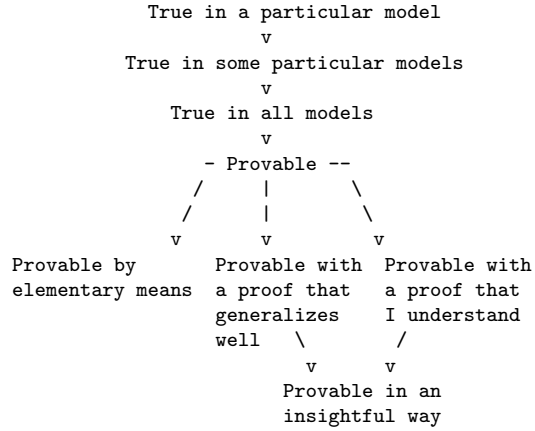
More notes about archetypes (1)

There are different levels of mathematical truths -
some ideas are "obvious", and I can use them implicitly
or by just mentioning them briefly, and my interlocutor
will accept them; other ideas - for example, theorems
that have been published, but whose proofs are hard -
I can only use with extreme care, and a proof that makes
use of these hard theorems automatically gets marked
with a small red or black flag -

A statement like "the composite (a,b)|->(b,a)|->(a,b)
is the identity" takes very little "mental space", and
is intuitively true, even though in different settings
it may have different proofs, and some of these proofs -
in λ-calculus, for example - may look too technical at
first...

Where do these "intuitively true statements" live?
What languages can we develop to work with them?

Here is a diagram showing some kinds of mathematical
truths. The arrows go from "weaker truths" to "stronger
truths".



More notes about archetypes (2)

(Note: D stands for a DAG - usually finite,
and in most examples having at most five
vertices)

For me, this is the archetype behind the idea of "topos":

  Every Set^D is a topos.

However, the resulting concept is much more general than that -
and the "archetypical language" for a topos comes from a very
particular case - Set^D = Set, D = the one-point DAG.

This is the archetype of a "sheaf":
Every Set^O(D)^op has a notion of ‘‘\dbul-sheaf’’.
The resulting notion of ‘‘sheaf’’ - based on Lawvere-Tierney
topologies and modalities - admits several other kinds of
sheaves: ‘‘¬¬-sheaves’’, ‘‘(\aa⊃)-sheaves’’, ‘‘(\bb∨)-sheaves’’,
and forcing.

The precise definition of what a topos is
can be expressed in the language of the archetype;
we can use the language of the archetype to express most
(i.e., "enough") of the most important constructions
and theorems; we can use it to help us build dictionaries
between the standard, more precise notations; and proofs done
in the archetype take relatively little mental space, and
generalize easily. Also, the archetypal language for a
topos may collapse (in the syntax!) some things that are
different in some toposes, but it is "non-trivial enough" -
it does not collapse too much.

More notes about archetypes (3)

```
What do I expect from an archetype and its language?

"It should be expressive enough"
  Most of the basic theory - constructions and theorems -
  can be done in the archetypal language, and then lifted
  to the general case; the general definition can be
  expressed using the archetypal language.

"It collapses enough"
  The archetypal syntax identifies (in the "wd[·]" sense)
  many constructions that are expected to give the same
  results.

"It is non-trivial enough"
  Even though the archetypal syntax may identify some
  constructions that are distinct in some models
  (see the example "x,ab,cd<-|x,a,b,c,d" in monads)
  it does not collapse too much.

Also, it can't take up too much "mental space" -
```